ARCHI2 - Compte-rendu du TME3

Nicolas Phan

pour le 2 Mars 2018

Table des matières

1 Application logicielle

1.1 Question C1

L'instruction `.word main` place le code commencant au label `main` avant les tableaux A, B et C, le code de la fonction main commence donc au début du segment `CODE`, autrement dit à l'adresse `0x00400000`.

Le début de boucle commencce quatre instructions après le début de main, donc 4 mots plus tard, donc @boucle = @main + 4×4 = `0x00400010`

1.2 Question C2

Les directives `.word main` et `.space 124` indiquent que le le tableau A se situe 4(taille du word) + 124 octets plus loin que le début du segment DATA, donc :

@base 1 = `SEG_DATA_BASE` + 128 = `0x01000080`

Puis entre A et B il y a 20 mots plus un espace de 48 octets, d'où :

@base 2 = @base 1 + $4 \times 20 + 48$ = `0x01000100`

De la même manière :

@base 3 = @base 2 + $4 \times 20 + 48$ = `0x01000180`

1.3 Question C3

Placer `sw` après `bne` est une optimisation de code permettant de réduire le nombre de cycles de gel par itérations. Apres le `bne` l y aura un Delayed Slot, autant de pas y mettre un `Nop` mais une instruction utile.

1.4 Question C4

Ici :

$$\begin{aligned} \text{nombre cycles par iteration} &= \text{nb instructions} + \text{nb cycles gel} \\ &= 7 + 0 \\ &= 7\text{cycles/it} \end{aligned}$$

2 Fonctionnement du cache instruction

2.1 Question D1

Les lignes de cache mesurent 4 mots = 16 octets = 2^4 octets donc le champ `BYTE` comporte 4 bits.

Une ligne de cache mesure 16 octets et le cache est de 128 octets, il comporte donc $\frac{128}{16} = 8$ cases. Le cache comporte 8 cases donc 8 ensemble (direct mapping) donc le champ `SET` comporte 3 bits.

Par soustraction, le champ `TAG` comporte $32 - 4 - 3 = 25$ bits

Pour l'adresse `0x00400000` , nous avons :

`BYTE = 0`

`SET = 0`

`TAG = 0000 0000 0100 0000 0000 0000 0 = 0x8000`

2.2 Question D2

La toute première ligne lui produit un miss compulsif, elle correspond à l'adresse `0x00400000` (`SET` = 0, `TAG` = `0x8000`) donc le controlleur de cache ramène la ligne de cache comportant les 4 premières instructions dans la case 0 du cache.

Les 4 premières instructions sont exécutées puis la lecture de la 5ème produit un miss compulsif, cette instruction à l'adresse `0x00400010` (`SET` = 1, `TAG` = `0x8000`) donc la ligne de cache correspondante sera ramenée à la case 1.

Par cette logique de quatre en quatre, ce seront donc les instructions `lui` , `lw $10` et `add` qui entraineront un miss sur le cache d'instructions.

Etat du cache d'instructions à la fin de la première itération :

CASE	TAG	V	WORD3	WORD2	WORD1	WORD0
0	`0x8000`	1	`li`	`li`	`addiu`	`lui`
1	`0x8000`	1	`addi`	`addi`	`lw`	`lw`
2	`0x8000`	1	`lui`	`sw`	`bne`	`add`
3		0				
4		0				
5		0				
6		0				
7		0				

TABLE 1 – Etat du cache d'instructions

2.3 Question D3

L'exécutions des itérations suivantes ne changent pas l'état du cache d'instructions car toutes les instructions à exécuter sont déjà présentes dans le cache d'instructions. A la fin de la 20ème itération, l'état du cache est le même qu'à la fin de la première.

L'exécution de la boucle entraine donc 2 miss (pour la première itération) pour l'exécution de $20 \times 2 = 40$ instructions de lecture. Le taux de miss est donc environ de $\frac{2}{40} = 5$ %.

2.4 Question D4

L'état `MISS_SELECT` est indispensable pour les caches de niveau d'associativité 2 où plus. Dans ces caches, plusieurs cases de cache sont possibles pour stocker une ligne de cache donnée, donc l'automate doit nécessairement choisir la case où stocker une ligne.

2.5 Question D5

Etat	**Signification**
IDLE	L'automate n'a affaire à aucun MISS
ERROR	Une adresse invalide a été demandée
MISS_SELECT	Un miss cachable s'est produit, l'automate sélectionne la case où stocker la ligne à ramener.
MISS_WAIT	Un miss cachable s'est produit, l'automate attend que l'automate PIBUS_FSM ait amenée la ligne de cache dans le registre tampon.
MISS_UPDT	L'automate envoie la ligne récupérée dans le registre tampon vers la case de cache choisie.
UNC_WAIT	Un miss non-cachable s'est produit, l'automate attend que PIBUS_FSM ait amenée la donnée demandée dans le registre tampon.
UNC_GO	Un miss non-cachable s'est produit, l'automate redirige la donnée recue dans le registre tampon vers le processeur.

TABLE 2 – Signification des états

Transition	Expression booleenne	Signification
C	$\overline{\texttt{IREQ}} + \overline{\texttt{IMISS}}$	Si le processeur ne demande rien ou s'il demande quelque chose qui est bien dans le cache, alors il n'y a pas miss.
A	IREQ . IMISS . IUNC	Le processeur demande une donnée non cachable, cela a (nécéssairement) entrainé un miss.
B	IREQ . IMISS . $\overline{\texttt{IUNC}}$	Le processeur demande une donnée cachable mais celle-ci n'est pas dans le cache.
I	1	
F	VALID + $\overline{\texttt{ERROR}}$	L'automate a recu une réponse sans erreur de la part de PIBUS_FSM, il passe à la mise à jour du cache.
G	VALID +ERROR	L'automate a recu une réponse d'erreur de la part de PIBUS_FSM, il passe à l'état d'erreur.
H	$\overline{\texttt{VALID}}$	PIBUS_FSM n'a toujours pas donné sa réponse, l'automate continue d'attendre.
N	1	
J	$\overline{\texttt{VALID}}$	PIBUS_FSM n'a toujours pas donné sa réponse, l'automate continue d'attendre.
K	VALID +ERROR	L'automate a recu une réponse d'erreur de la part de PIBUS_FSM, il passe à l'état d'erreur.
L	VALID + $\overline{\texttt{ERROR}}$	L'automate a recu une réponse sans erreur de la part de PIBUS_FSM, il passe à l'état de redirection de la donnée vers le processeur.
M	1	

TABLE 3 – Transitions de l'automate

2.6 Question D6

L'activation du signal RESETN force cet automate anisi que les deux autres à l'état IDLE.

3 Fonctionnement du cache de donnees

3.1 Question E1

Pour l'adresse `0x01000080` (`A[0]`), nous avons :

`BYTE = 0`

`SET = 0`

`TAG = 0000 0001 0000 0000 0000 0000 1 = 0x20001`

Pour l'adresse `0x01000100` (`B[0]`), nous avons :

`BYTE = 0`

`SET = 0`

`TAG = 0000 0001 0000 0000 0000 0001 0 = 0x20002`

Le premier `lw` entraine un miss compulsif où les 4 premieres cases du tableau A sont ramenées dans la case 0 du cache. Puis le deuxième `lw` entraine aussi un miss compulsif, sauf que la case associée aux 4 premiers mots du tableau B est aussi la case 0, donc le début du tableau A est évincé du cache.

Etat du cache de données à la fin de la première itération :

CASE	TAG	V	WORD3	WORD2	WORD1	WORD0
0	0x20002	1	104	103	102	101
1		0				
2		0				
3		0				
4		0				
5		0				
6		0				
7		0				

TABLE 4 – Etat du cache de données

3.2 Question E2

A la deuxième itération, le premier lw entrainera un miss conflit cette fois, car il a été évincé précédemment par le tableau B : Les deux tableaux sont en conflit donc lorsque les lw n'entrainent pas de miss compulsifs, ils entrainent toujours des miss conflit. Le taux de miss est donc de 100 %.

Les miss seront compulsifs aux itétations n où $n \mod 4 = 1$, là où la case du tableau demandée est au début d'une nouvelle ligne. Dans les autres cas il s'agira de miss conflits.

Etat du cache de données à la fin de la dernière itération :

CASE	TAG	V	WORD3	WORD2	WORD1	WORD0
0	0x20002	1	104	103	102	101
1	0x20002	1	108	107	106	105
2	0x20002	1	112	111	110	109
3	0x20002	1	116	115	114	113
4	0x20002	1	120	119	118	117
5		0				
6		0				
7		0				

TABLE 5 – Etat du cache de données

3.3 Question E3

Transition	Expression booleenne	Signification
C	$\overline{\texttt{DREQ}} + (\overline{\texttt{WRITE}} \cdot \overline{\texttt{DMISS}})$	Si le processeur ne demande rien ou s'il demande à lire quelque chose qui est bien dans le cache, alors il n'y a pas miss.
A	$\texttt{DREQ} \cdot \overline{\texttt{WRITE}} \cdot \texttt{DMISS} \cdot \texttt{DUNC}$	Le processeur demande à lire une donnée non cachable, cela a (nécéssairement) entrainé un miss.
B	$\texttt{DREQ} \cdot \overline{\texttt{WRITE}} \cdot \texttt{DMISS} \cdot \overline{\texttt{DUNC}}$	Le processeur demande à lire une donnée cachable mais celle-ci n'est pas dans le cache.
D	$\texttt{DREQ} \cdot \texttt{WRITE} \cdot \overline{\texttt{DMISS}}$	Le processeur demande à écrire à une adresse présente dans le cache, l'automate passe à l'état de mise à jour du cache
E	$\texttt{DREQ} \cdot \texttt{WRITE} \cdot \texttt{DMISS}$	Le processeur demande à écrire à une adresse absente du cache, l'automate n'a pas à mettre à jour le cache, il passe directement à l'état où il poste cette requete dans le TEP.
P	1	L'automate a fini de mettre à jour le cache suite à une demande d'écriture, il passe donc à l'état où il poste cette requête dans le TEP.
I	1	
F	$\texttt{VALID} + \overline{\texttt{ERROR}}$	L'automate a recu une réponse sans erreur de la part de PIBUS_FSM, il passe à la mise à jour du cache.
G	$\texttt{VALID} + \texttt{ERROR}$	L'automate a recu une réponse d'erreur de la part de PIBUS_FSM, il passe à l'état d'erreur.
H	$\overline{\texttt{VALID}}$	PIBUS_FSM n'a toujours pas donné sa réponse, l'automate continue d'attendre.
N	1	
J	$\overline{\texttt{VALID}}$	PIBUS_FSM n'a toujours pas donné sa réponse, l'automate continue d'attendre.
K	$\texttt{VALID} + \texttt{ERROR}$	L'automate a recu une réponse d'erreur de la part de PIBUS_FSM, il passe à l'état d'erreur.
L	$\texttt{VALID} + \overline{\texttt{ERROR}}$	L'automate a recu une réponse sans erreur de la part de PIBUS_FSM, il passe à l'état de redirection de la donnée vers le processeur.
M	1	

TABLE 6 – Transitions de l'automate

4 Accès au PIBUS

F1] Supposons que le processeur fasse une demande de lecture alors qu'il y a des requêtes d'écriture dans le TEP. Pour peu que la lecture demandée concerne la même adresse qu'une écriture postée, répondre à la demande de lecture en premier entrainerait une inconsistence mémoire, la donnée renvoyée au processeur serait obsolète. Une manière de remédier à cela est de comparer les adresses des requêtes d'écritures de la TEP avec l'adresse de la lecture demandée, mais cela apporte des complication en terme de conception.

Une autre solution, plus simple mais plus radicale, consiste à toujours effectuer toutes les écritures dans le TEP avant d'effectuer quelconque lecture. De cette manière, nous sommes assurés que la consistence mémoire est respectée, toutes les données renvoyées au processeur seront bien à jour. C'est la solution retenue ici, les écritures sont donc toujours prioritaires.

L'inconvénient est que si le TEP est rempli, une demande de lecture entrainera beaucoup de cycles de gel car la demande sera satisfaite qu'une fois que TOUTES les écritures du TEP auront été effectuées, et tant que la lecture n'est pas terminée le processeur reste gelé. Le processeur gèlera donc le temps d'effectuer les N écritures postées plus la lecture en question.

F2] Le mécanisme utlisé par les automates clients pour transmettre une requête à l'automate serveur est le mécanisme de tampon protégé par une bascule RS, c'est le mécanisme le plus simple et le plus utilisé pour la communication entre automates.

Ce mécanisme consiste en un registre tampon, ici ADDR de 28 bits (et non pas 32 car les adresses fournies sont nécéssairement alignées en mémoire, et étant donné que les lignes de caches font 2^4 octets, les 4 LSB des adresses sont nuls), c'est l'adresse fournie par le client au serveur. La deuxième partie du mécanisme est la bascule RS (ici REQ) qui est un automate à deux états indiquant si une requête est en suspens et dans le même temps, détermine qui a la parole (client ou serveur).

Lorsque REQ est à 0, aucune requête n'est en suspens, le client a la parole et la prendra lorsqu'il aura une requête à soumettre, auquel cas il peut mettre REQ à 1. Le serveur, quant à lui, ne peut pas modifier REQ.

Lorsque REQ est à 1, une requête est en cours de traitement, le serveur a la parole et la prendra lorsqu'il aura satisfait la requête, auquel cas il mettra REQ à 0. Le client, quand à lui, ne peut pas modifier REQ.

Pour transmettre la réponse du serveur aux clients, le même mécanisme est employé. Le signal en sortie de la bascule RS est IRSP et vaut 1 lorsqu'une réponse du serveur est disponible et attent d'être récupérée par le client, c'est au client de remettre IRSP à 0 lorsqu'il récupère la réponse. IRSP vaut 0 lorsqu'aucune réponse n'est disponible, c'est au serveur de mettre à 1 lorsqu'il en délivrera une.

F4]